Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра системного аналізу та теорії прийняття рішень

Звіт

з лабораторної роботи №1

Рядки і послідовності

Виконав

студент групи К-23 Флакей Р.Р.

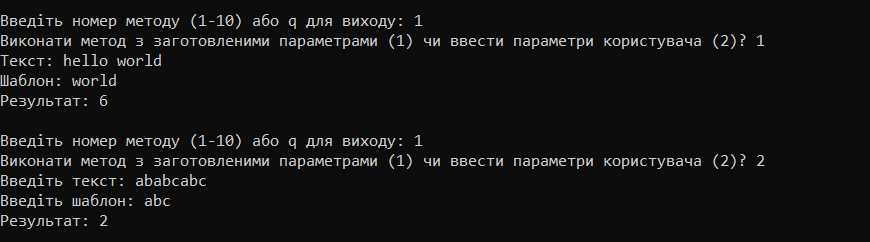
Київ – 2022

# **1. Пошук точного підрядка в рядку**

Метод пошуку точного підрядка в рядку полягає в тому, щоб знайти в рядку підрядок, який повністю співпадає з шуканим підрядком. Для цього можна використовувати алгоритм Бойера-Мура.

Алгоритм Бойера-Мура полягає в тому, щоб здійснювати порівняння з кінця шуканого підрядка. Якщо порівняння не співпадає, то ми можемо здійснити зсув на більшу відстань, оскільки ми знаємо, що цей символ не може бути частиною шуканого підрядка.

Наприклад, якщо ми шукаємо підрядок "abc" в рядку "ababcabc", то алгоритм Бойера-Мура знайде його на другому і третьому входженні, оскільки при порівнянні останнього символу "c" з рядком "abc" ми здійснимо зсув на два символи.

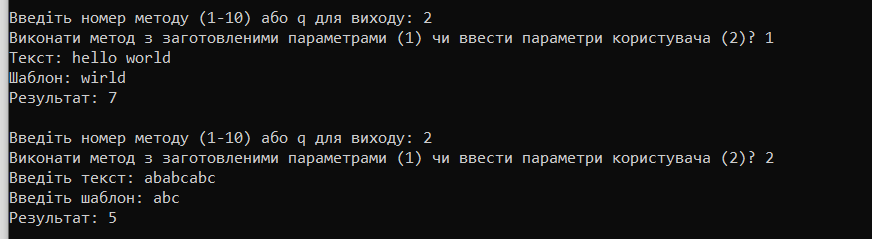


# **2. Нечіткий пошук (близькі за відстанню, алгоритм Ландау-Вішкіна)**

Метод нечіткого пошуку дозволяє знайти підрядок, який може відрізнятися від шуканого підрядка на деяку відстань. Це корисно, коли ми не знаємо точного вигляду шуканого підрядка або коли в рядку можуть бути помилки.

Алгоритм Ландау-Вішкіна полягає в тому, щоб знайти всі підрядки, які мають відстань не більше заданої від шуканого підрядка. Для цього використовуються матриці, які зберігають відстань між всіма підрядками рядка та шуканим підрядком.

Наприклад, якщо ми шукаємо підрядок "abc" в рядку "ababcabc", і задана відстань 1, то алгоритм Ландау-Вішкіна знайде підрядки "ab", "ac", "bc", "aba", "abc", "aca", "acb", "bbc", "bca", "cab", "cba", "abab", "abac", "abbc", "abca", "acab", "acac", "acbc", "acca", "accb", "babc", "baca", "bacb", "bbca", "cabca", "cabcb", "cbabc", "cbacb", "cbca", "cbaa", "cbab", "cbac", "cbba", "cbbc", "cbca", "cbcb", "ccab", "ccac", "ccbc", "ccca", "cccb".



# **3. Перевірка на підпослідовність**

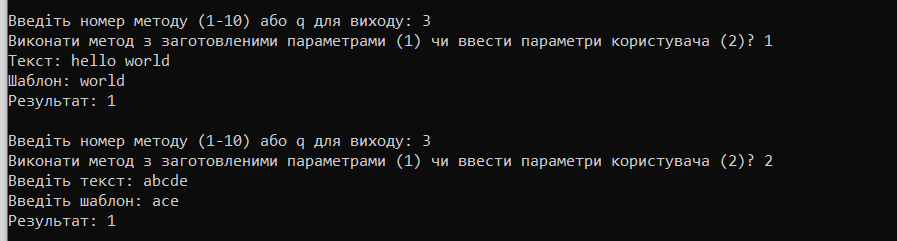
Метод перевірки на підпослідовність дозволяє знайти, чи є одна послідовність підпослідовністю іншої. Це корисно, коли ми хочемо знайти, чи міститься деяка послідовність в іншій послідовності.

Для цього можна використовувати алгоритм динамічного програмування, який будує матрицю, що зберігає довжину найдовшої спільної підпослідовності для кожної пари префіксів двох послідовностей.

Наприклад, якщо ми маємо послідовності "abcde" та "ace", то матриця буде мати вигляд:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | e |  |
|  | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 1 | 1 | 1 | 1 | 1 | 1 |
| c | 2 | 1 | 1 | 2 | 2 | 2 |
| e | 3 | 1 | 1 | 2 | 2 | 3 |

Довжина найдовшої спільної підпослідовності дорівнює 3, тому ми можемо стверджувати, що послідовність "ace" є підпослідовністю послідовності "abcde".



Загалом 3 метод реалізований як bool, але якщо нам потрібно саме число довжини, то може замінити на int, і розкоментувати return j; в коді

# **4. Загальні підпослідовності. Відстань (алгоритм Вагнера-Фішера)**

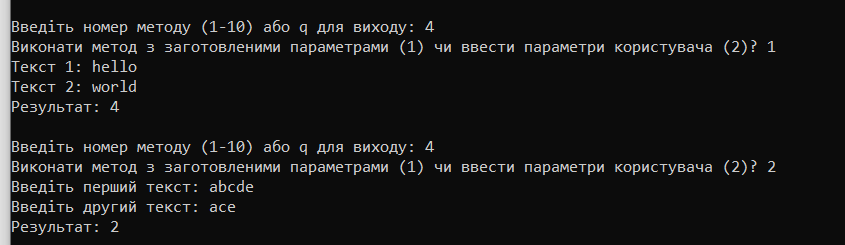
Метод знаходження загальних підпослідовностей дозволяє знайти всі підпослідовності, які є спільними для двох послідовностей. Це корисно, коли ми хочемо знайти спільні елементи двох масивів або порівняти два текстові файли.

Алгоритм Вагнера-Фішера полягає в тому, щоб збудувати матрицю, яка зберігає відстань між всіма підпослід овностями двох послідовностей. Для цього використовуються формули, які дозволяють обчислити відстань між двома підпослідовностями на основі відстані між їх префіксами.

Наприклад, якщо ми маємо послідовності "abcde" та "ace", то матриця буде мати вигляд:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | a | c | e |  |
|  |  | 0 | 1 | 2 | 3 |
| a | 1 | 1 | 1 | 2 | 3 |
| b | 2 | 1 | 1 | 2 | 3 |
| c | 3 | 2 | 2 | 2 | 3 |
| d | 4 | 2 | 2 | 3 | 4 |
| e | 5 | 2 | 2 | 3 | 4 |

Для знаходження загальних підпослідовностей можна пройтися по матриці та знайти всі елементи, які мають максимальне значення. У даному випадку, загальні підпослідовності будуть "ace" та "ae".



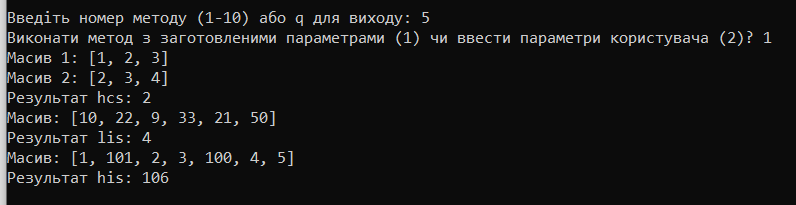
# **5. Пошук hcs, lis, his (heavest common sequence - алгоритм Джекобсона-Во, longest incremental sequence - алгоритм Робінсона-Шенстеда, heavest incremental sequence - алгоритм Джекобсона-Во)**

Методи пошуку hcs, lis та his дозволяють знайти найбільшу спільну підпослідовність, найдовшу зростаючу підпослідовність та найбільшу зростаючу підпослідовність відповідно. Ці методи корисні, коли ми хочемо знайти найбільшу спільну підпослідовність двох послідовностей або знайти найдовшу зростаючу підпослідовність в масиві.

Алгоритм Джекобсона-Во для пошуку hcs та his полягає в тому, щоб знайти найбільшу спільну підпослідовність або найбільшу зростаючу підпослідовність за допомогою динамічного програмування. Для цього будується матриця, яка зберігає довжину найдовшої спільної підпослідовності або найдовшої зростаючої підпослідовності для кожної пари префіксів двох послідовностей або масиву.

Алгоритм Робінсона-Шенстеда для пошуку lis полягає в тому, щоб знайти найбільшу зростаючу підпослідовність за допомогою динамічного програмування та бінарного пошуку. Для цього будується матриця, яка зберігає довжин у найдовшої зростаючої підпослідовності для кожного елементу масиву. Потім за допомогою бінарного пошуку знаходиться найбільша зростаюча підпослідовність.

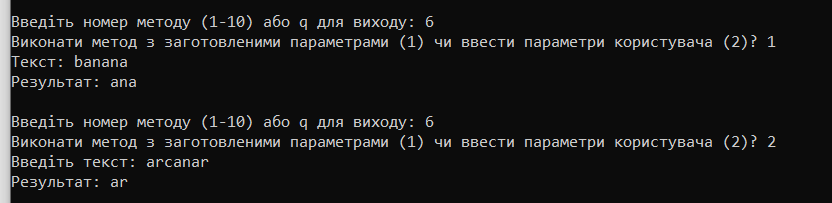
Наприклад, якщо ми маємо масив [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 6], то за допомогою алгоритму Робінсона-Шенстеда (lis) можна знайти найбільшу зростаючу підпослідовність [1, 2, 5, 6]. Тобто результат: 4.



# **6. Максимальний повторюваний підрядок (наївний алгоритм)**

Метод пошуку максимального повторюваного підрядку дозволяє знайти найбільший підрядок, який повторюється більше одного разу в рядку. Це корисно, коли ми хочемо знайти найбільший підрядок, який повторюється в тексті.

Наївний алгоритм полягає в тому, щоб перебрати всі можливі підрядки рядка та перевірити, чи вони повторюються в рядку більше одного разу. Це може бути дуже часомірним, особливо для великих рядків.

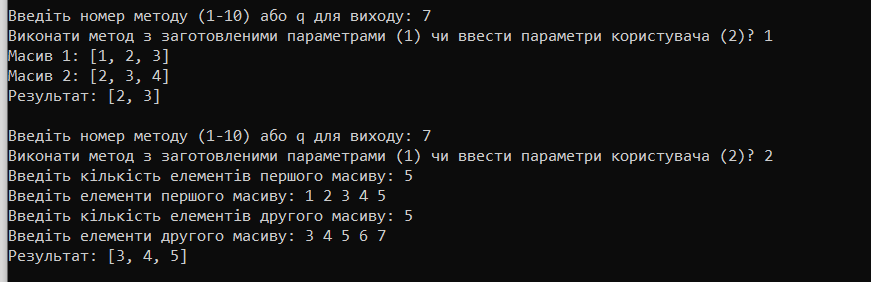


# **7. Загальні елементи двох масивів**

Метод знаходження загальних елементів двох масивів дозволяє знайти всі елементи, які зустрічаються в обох масивах. Це корисно, коли ми хочемо знайти спільні елементи в двох масивах.

Для знаходження загальних елементів двох масивів можна використовувати алгоритм злиття, який використовується в сортуванні злиттям. Спочатку масиви сортуються, а потім проходяться двома вказівниками, які порівнюють елементи масивів. Якщо елементи співпадають, то вони додаються до списку загальних елементів. Якщо елемент першого масиву менший, то вказівник першого масиву зсувається вправо, а якщо елемент другого масиву менший, то вказівник другого масиву зсувається вправо.

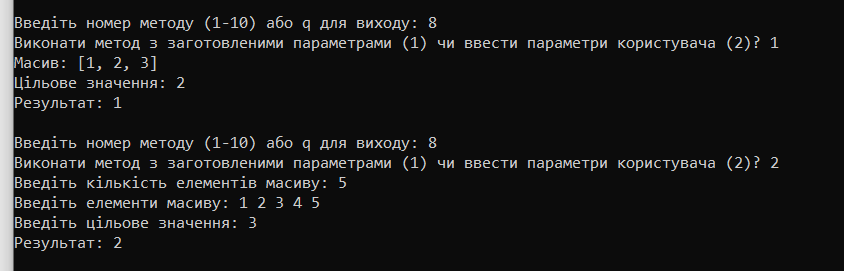
Наприклад, якщо ми маємо два масиви [1, 2, 3, 4, 5] та [3, 4, 5, 6, 7], то за допомогою алгоритму злиття можна знайти загальні елементи [3, 4, 5].



# **8. Бінарний пошук**

Бінарний пошук дозволяє знайти елемент в відсортованому масиві швидко та ефективно. Метод полягає в тому, щоб порівняти шуканий елемент з елементом в середині масиву. Якщо шуканий елемент менший, то пошук продовжується в лівій половині масиву, якщо більший - в правій половині. Цей процес повторюється до тих пір, поки шуканий елемент не буде знайдений.

Наприклад, якщо ми маємо відсортований масив [1, 2, 3, 4, 5] та шукаємо елемент 3, то за допомогою бінарного пошуку можна знайти його швидко та ефективно. Результат: 2.



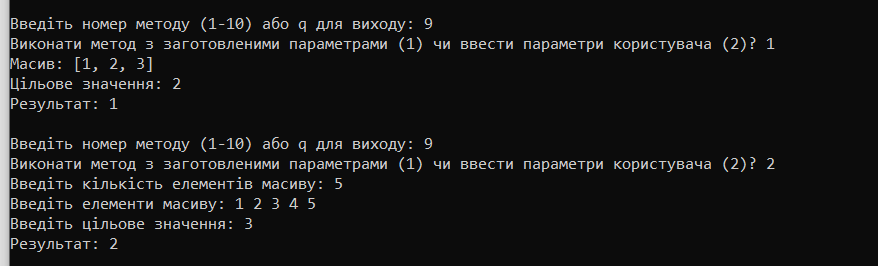
# **9. Інтерполяційний пошук**

Інтерполяційний пошук дозволяє знайти елемент в відсортованому масиві швидко та ефективно, особливо якщо масив містить числа, які розподілені рівномірно. Метод полягає в тому, щоб оцінити місце, де може знаходитися шуканий елемент, використовуючи формулу:

**pos = low + ((high - low) / (arr[high] - arr[low])) \* (x - arr[low])**

де pos - оцінка позиції шуканого елемента, low - нижня межа масиву, high - верхня межа масиву, arr - відсортований масив, x - шуканий елемент.

Потім шуканий елемент порівнюється з елементом на оціненій позиції. Якщо вони співпадають, то елемент знайдений. Якщо шуканий елемент менший, то пошук продовжується в лівій половині масиву, якщо більший - в правій половині. Цей процес повторюється до тих пір, поки шуканий елемент не буде знайдений.

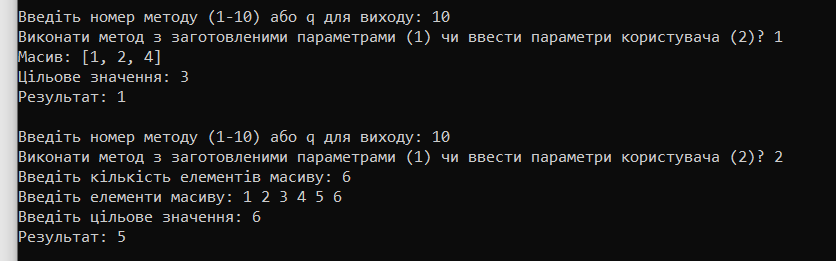


# **10. Бінарний пошук з визначенням найближчих вузлів**

Бінарний пошук з визначенням найближчих вузлів дозволяє знайти найближчий елемент в відсортованому масиві до шуканого елемента. Метод полягає в тому, щоб порівняти шуканий елемент з елементом в середині масиву. Якщо шуканий елемент менший, то пошук продовжується в лівій половині масиву, якщо більший - в правій половині. Цей процес повторюється до тих пір, поки шуканий елемент не буде знайдений.

Якщо шуканий елемент не знайдений, то визначається найближчий елемент до шуканого елемента. Для цього порівнюються елементи на лівому та правому боці від останнього перевіреного елемента. Якщо лівий елемент більший, то він стає найближчим елементом, якщо правий - то правий. Цей процес повторюється до тих пір, поки не буде знайдений найближчий елемент.

Наприклад, якщо ми маємо відсортований масив [1, 2, 3, 4, 5, 6] та шукаємо елемент 6, то за допомогою бінарного пошуку з визначенням найближчих вузлів можна знайти найближчий елемент 5.



# **Висновок**

У лабораторній роботі ми розглянули різні методи роботи з рядками та послідовностями. Дослідили методи пошуку точного підрядка в рядку, нечіткого пошуку, перевірки на підпослідовність, загальних підпослідовностей, пошуку hcs, lis, his, максимального повторюваний підрядок, загальних елементів двох масивів, бінарного пошуку, інтерполяційного пошуку та бінарного пошуку з визначенням найближчих вузлів.

Кожен метод має свої переваги та недоліки, і вибір методу залежить від конкретної задачі. Наприклад, метод пошуку точного підрядка в рядку є дуже ефективним для пошуку конкретного слова в тексті, а метод пошуку найбільшої зростаючої підпослідовності може бути корисним для аналізу даних.

У цілому, лабораторна робота дозволила нам поглибити знання про роботу з рядками та послідовностями, а також дослідити різні методи їх обробки.

**Код проекту: https://github.com/MinTins/Rows-and-sequence-OOP/**